Mouseketool

Authors: Rosina Ananth, Saumya Bhandarkar, Sarah Gaiter
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of wirelessly controlling a mouse through a dedicated fail-safe sensor and common keystrokes on a laptop screen from afar. Improving upon existing technology such as computer mice and remotes, the Mouseketool leverages variable resistive sensors, an Inertial Measurement Unit utilizing the Kalman filter, and Bluetooth to create a seamless experience and a rich gesture language for users to move, click, and perform other keyboard actions. The project aims to improve the experience of human-computer interactions.**

***Index Terms*— Wireless mouse, Inertial Measurement Unit (IMU), Human-Computer Interaction, Bluetooth Low Energy, GUI, Wearable Technology, Embedded Systems, Touch Sensors, Flex Sensors, Analog-Digital Converter**

1 INTRODUCTION

The field of human-computer interaction is constantly evolving and is a critical part of our lives. Traditional input devices like the mouse, keyboard, and remote are fundamental in how we interact with computers, but in terms of accessibility, can be expanded on. Our product, the Mouseketool, offers a solution that is more intuitive, versatile, and accessible to interact with a computer. The Mouseketool is essentially a glove embedded with sensors that converts motion to mouse movements and touch to keystrokes. Traditional input devices like a mouse can cause discomfort with prolonged use, but the Mouseketool eliminates the need for constant wrist and hand movements, offering a comfortable and ergonomic alternative. It can also be accessible and convenient for those with physical disabilities involving limited mobility. For certain applications requiring precise control, the Mouseketool can increase efficiency and productivity, such as design, gaming, and 3D modeling.

2 USE-CASE REQUIREMENTS

2.1 Accuracy:

We break down accuracy into two components: user experience of accuracy and technical accuracy. For user experience, we aim to have a test group of users rate our devices an average score of 90% with regards to its usability and how well they perceive it to pick up their movements. In terms of technical accuracy, we aim for our product to recognize and carry out our gesture language with 90% accuracy.

2.2 Weight:

We aim to have our product weight 113-170 grams (4-6 ounces), the weight of an average sports watch. The weight of the product is crucial to ensuring user comfort and minimizing physical strain during prolonged use. By targeting this weight range, our product will be lightweight and unobtrusive, preventing users from experiencing discomfort or fatigue, as proven by those that wear sport watches.

2.3 Latency:

The product should exhibit minimal input-to-response latency, with a maximum acceptable latency of 300 milliseconds. Low latency is an important factor for applications where real-time interaction is required, such as gaming and virtual reality. High latency can negatively impact user experience, leading to reduced usability. Therefore, minimizing latency is essential to providing a seamless user experience.

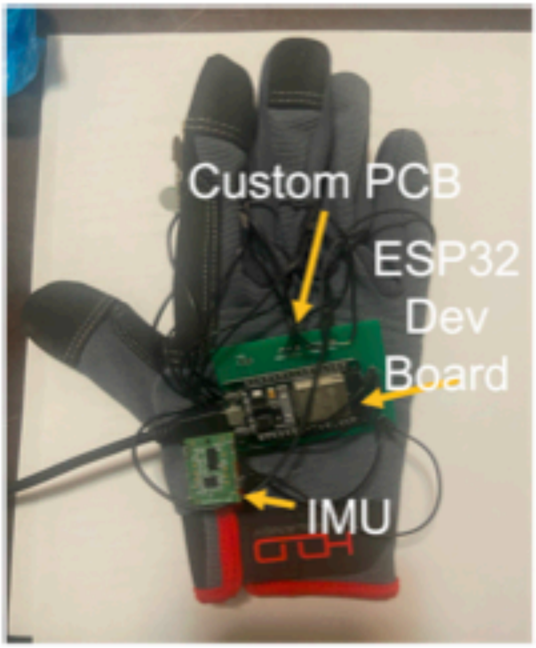
2.4 Wireless Range:

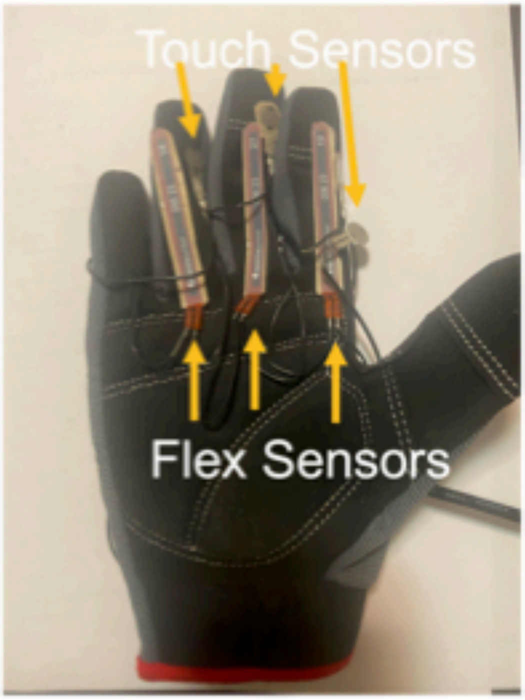
We aim to have our product work up to a range of 2.28 meters (7.5 ft). We selected this range because it is the average range of a wireless game controller. This range is selected to provide users with sufficient mobility and freedom to interact with their computer without having to stay in a specific location. This allows users to sit comfortably on a couch or move around in their environment, similar to the average viewing distance from a television.

2.5 Battery Life:

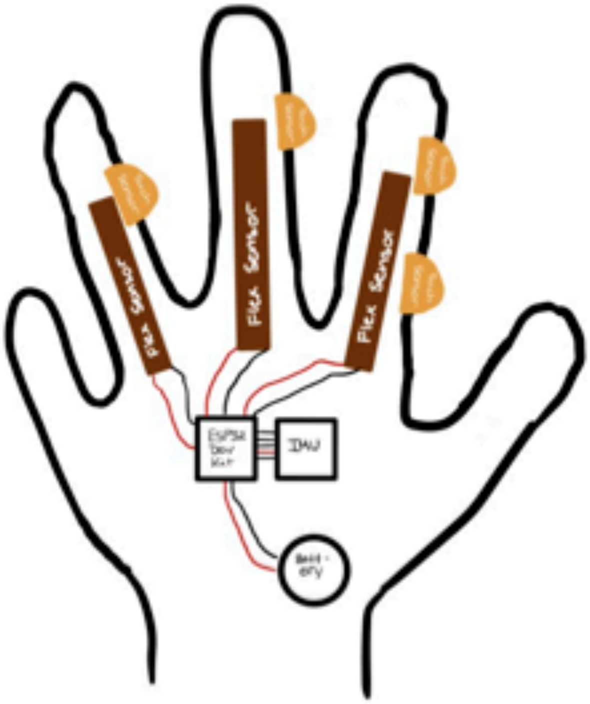
We selected the battery life of the product to be 2-3 hours. This is so users can use the product for extended periods without frequent recharging. This is important for applications that require prolonged periods such as watching movies and doing work. A longer battery life requires a large battery, and we did not want to sacrifice the weight of the product. Thus, we chose a reasonable battery life of 2-3 hours.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION





Our embedded system is made up of a microcontroller and several sensors (variable resistors and an inertial measurement unit (IMU)) which are spread across a glove. The touch sensors, or force-sensitive resistors, are placed on the fingertips of the thumb, pointer, middle, and ring fingers as well as on the side of the pointer finger. The flex sensors are placed on the pointer, middle, and ring fingers to detect bends in the fingers. Finally, the IMU and the microcontroller are placed on the back of the hand of the glove, mounted on a PCB to allow the user's hand to move as freely as possible. A diagram of the sensor layout can be seen below.



The Mouseketool should only carry out mouse and keyboard actions when the user intends to make these actions. Thus, we designed our system to have a fail-safe trigger. The IMU will only register and send data when the sensor on the side of the index finger is triggered. We have similar failsafe triggers for the touch and flex sensors. For the touch sensors on the tips of fingers, the system will only register a finger press if both the thumb and the finger are pressed at the same time. For the flex sensors, the system will only register a finger flex if the pad of the respective finger is also pressed into the palm of the hand.

As per the Bluetooth architecture plan (see Fig. 5), it's important to note that Bluetooth Low Energy (BLE) doesn't follow the traditional guidelines of a client-server model. Rather, we made the ESP32 the server, and the Python code the client. As mentioned before, we have a fail-safe for the mouse located on the side of the index finger. When this is triggered, the board will continuously transmit linear acceleration on the x and z axes via Bluetooth notification to the Python script every 50ms. On the client side, callback notifications are received, and the code calculates mouse movements using a double integral formula with bias-correction. It then uses the python built in Pyautogui to move the mouse on your screen. If the failsafe is not detected, the board will read in the values of the other sensors on the glove (touch and flex sensors) and send out an opcode to be transmitted. Again, on the client side, when callback notifications are received, it decodes the sent opcode and carries out the customized gesture. The Arduino code running on the esp works as follows. First, the esp32 initializes both the Bluetooth device and its various sensors. The esp polls its sensors every 30ms to ensure response rate is as quick as possible. When a gesture is detected, the esp sends a notification out and ignores all other gestures and sensor inputs for 1.5 seconds. We added this delay as it ensures that a single gesture isn't transmitted repeatedly, which would be inconvenient to users.

As for the IMU diagram (see Fig. 6), on the board side, once the IMU is moved and the failsafe triggered, the board will begin sending over acceleration data to a python receiver for processing. From here, the raw data will go through a Kalaman filter to reduce compounding error over time from the integration. After this, we will double integrate the acceleration to get the position of the glove relative to its starting location, and use PyAutoGUI to move the mouse to the new position. We used a 3-state Kalman filter, one for position, velocity, and acceleration to correct for the error introduced by double integration and measurement noise. We iterated on the design, selecting different values for the transition covariance and initial state covariance based on the IMU values given. We specifically chose a low covariance for acceleration since our IMU is pretty accurate in its calculations, and higher values for the other two states, so the filter is stiffer, and corrects these values more over time.

As per the hardware architecture plan (see Fig. 4), the touch and flex sensors interact with the main development board through analog inputs. Each sensor will receive power (3.3V) from the board and feed information to the ESP32. The IMU will communicate with the development board via I2C, and will also receive power (3.3V) from the Vcc pin on the board. In order to supply power to the system and peripherals, we will have a battery ranging from 3.7V to 5V.

Some components of our system have changed since our design stage. The first change made was the placement of the touch sensors on the glove. When we first designed the placement of each sensor on the glove, we positioned each touch sensor at the tip of each finger and each flex sensor relatively high, right below the touch sensors. After a small user study of about 5 people, our accuracy was very low, with about 40% of the sensors working as promised. We saw that as people were reaching to touch the touch sensors, their fingers were bending just enough to trigger the flex sensors as well. We then decided to move the touch sensors down and to the side of their original positions, so the users won't have to bend their fingers as much to reach the touch sensors. As a quick demonstration, when you touch your fingertips with your thumb, you can see that your finger bends quite significantly. Moving the sensors to the side and down allow you to touch each sensor without too much of a bend. With proper thresholding, these slight bends will not produce a trigger. With this new positioning, we were able to achieve our use case requirements and satisfy our users.

We also realized our IMU data needed additional processing than planned. This included feeding the processed data into a Kalman filter to mitigate compounding error, as well as adding interrupt functionality to the fail-safe sensor, to reduce undesired mouse drift. We implemented a 3-state Kalman filter for the x and z accelerations respectively, with the states being position, velocity, and acceleration. We modified the initial and transient state covariances to ensure that the filter would correct for position and velocity more harshly than the acceleration values. In addition, we reset velocity to zero and froze the mouse in place when our accelerometer measured several zero values for acceleration in a row, to avoid additional movement past what was gestured. For interrupt functionality, we introduced another Bluetooth notification opcode that would prevent mouse movements when the failsafe sensor wasn't being actively pressed.

4 DESIGN REQUIREMENTS

4.1 Battery Voltage:

Ideally, our system should be powered with a lithium ion battery that is no more than 5V. This upper limit will ensure that our system's battery life is long enough to meet our 2-3 hour requirement, but the weight does not go over our limit of 170 grams.

4.2 Bluetooth:

The Bluetooth Low Energy receiver should be able to sense the module from a distance of 2.28 meters away. This will ensure that the user can comfortably use our glove even when watching a movie on their TV from their couch.

5 DESIGN TRADE STUDIES

5.1 ESP32 Board vs. Nucleo

One of the first and most important design decisions we made was to use the ESP32S development board. This board has over 15 ADC channels, built-in bluetooth connectivity and Wi-Fi capability. Additionally, it comes in a small package of around 2 inches wide, has a low power mode, and is relatively cheap in cost. Some tradeoffs we were introduced to were that there isn't as much documentation for the development board itself. For now, we are basing most of our calculations off of the microcontroller datasheet, but certain things such as how the LDO on the board works, and the specifications for the board itself are a bit unknown. To get around this, we need to do extensive testing to find out the correct and operable thresholds for each pin and each voltage input. Overall, this board helps significantly with our weight, communication, and battery requirements. We were debating on using the STM32 Nucleo Board instead, mostly because of our familiarity with the board and its software from taking 18349, the multitude of ports and ADC channels, and the extensive documentation on the chip itself. However, by using the STM32

Nucleo, we would trade off size, which is an extremely important requirement for us to have. The Nucleo board's width ranges to around 82.5mm, which is much larger than the average size of a palm. Because we want our product to be usable, portable, and easy for users to consume, we decided to go with the ESP32, for the reasons listed above.

5.2 Sensors vs Computer Vision

When designing our system, we determined two main ways that we could recognize gestures: either recognizing physical movements using sensors (the IMU) attached to the hand or using computer vision. There were several tradeoffs we considered when making this design choice. We determined that implementing our faraway mouse with computer vision might result in a better user experience for users because there would be no need for them to wear bulky sensors. However, this would come at the cost of applicability - specifically, computer vision might fail or perform poorly in dimly lit environments or at further distances from the computer. Using physical sensors would allow users to use the product in a multitude of light conditions (ex. when users are watching a movie in a dark room) and at a further distance rather than having to be within range of a camera. Another cost was power. Running a CV module (especially with ML involved) would be very power intensive compared to our set of sensors. Finally, having a CV model deployed in the cloud or even a local model might result in a lot of unwanted latency for a mouse application, more so than the combination of sensors and Bluetooth. The sensor option, we determined, would allow us to process data at a low level with less latency, speeding up our overall pipeline.

5.3 Bluetooth vs USB vs WiFi

For our glove, we explored several protocols for sending data from the mouse to a receiver on the laptop. We determined that using USB would be the most reliable, with minimal latency incurred from networking and minimal connection interruptions. However, this would defeat the design purpose of having a mouse that can be used from a little over 2 meters away, since the USB would be inconveniently long. Between Bluetooth and WiFi, either protocol would have worked well for our design, especially since our microcontroller supported both natively. In the end, we determined that the power consumption of Bluetooth would be less than that of WiFi, since it is designed for lower energy usage. Thus, we settled on using Bluetooth to send data, with USB as our backup plan.

5.4 Sensors

When designing our glove layout, we had to decide how many sensors we wanted on our glove as well as the placement. Having more sensors would of course allow us to support more gestures and gather more data on hand movement. However, each sensor would consume some power, take up one of the limited ADC pins on our microcontroller, and possibly create some interference with the other sensors. For each fingertip, we decided to use force sensitive resistors. Some of the benefits of this include that they're relatively small and cheap, running around 1cm in diameter, and are effective variable resistors, so establishing thresholds to make the system more binary is trivial. However, some tradeoffs we face with this include their variance. Each force sensitive resistor has a different calibration of reading, so we need to calibrate each sensor individually with information on its ideal working state. For example, we noticed that some of the sensors, when not depressed at all, have very different readings, ranging from close to 0 to around 700 (after ADC calculations). We determined that for our touch resistors, having one on each finger and an additional one as the "trigger" for the IMU would be sufficient to allow for a rich gesture language while keeping the sensors far enough apart and the number of sensors below the number of available ADCs. For our flex sensors, we chose to place them only on 3 of the fingers rather than all of them. Again, this design decision was based on our choice to limit the number of sensors while keeping our language as rich as possible. We placed the flex sensors on the easiest-to-bend fingers to make it accessible for users. Again, each sensor is relatively small and cheap, so we will have to customize our calibration technique for each sensor individually.

5.5 PCB Mill vs Ordering Board

For mounting our components, we considered several approaches, balancing wearability and comfort with what would be most effective for connecting wires. We considered two options: ordering a custom PCB to mount our components to, and milling our own PCB in house. Even though milling was much cheaper and the lead time was much faster (around 10-15 minutes in Techspark), the board was too large to comfortably fit on the back of the glove. The machine we used was also unreliable and it was difficult to get a usably milled PCB without tens of iterations. Because of this, we decided to order a custom PCB online. Although it did cost a bit more than milling, we were able to use much smaller traces, and the traces were much cleaner than using a mill. This brought our final dimensions down more than 1" of area, to about 2x2.5".

6 SYSTEM IMPLEMENTATION

6.1 Hardware

The ESP32 Dev Kit has a multitude of ports for us to use for the sensors. As seen in the schematic pictured in Fig. 1, we can see that our board derives power from the VIN port on the board (this will be supplied by our battery), and supplies power to peripherals through the 3.3V VCC pin. From here, we use the GPIO pins that also function as ADC pins to get information from our touch sensors (TS) and flex sensors (FS), as shown in Fig. 2. Finally, our IMU communicates with the board via I2C, as seen in Fig. 3. We can see in our schematic that these wires are connected to the board in the dedicated I2C lines (D21 for SDA and D22 for SCL). We have indicated the use of 2.2K pullup resistors, which will either be soldered onto our main board, or configured internally on the board. Finally, the IMU requires 3 additional pins: reset (NRST), a host interrupt (HINTN), and a bootloader mode selection (BOOTN). Since our board doesn't have dedicated pins for reset, boot, and interrupt, we will simply create these functionalities in software, and essentially bitbang the pins.

6.2 Software

6.2.1 IMU

A critical component of this product is the inertial measurement unit. It is responsible for translating hand movement into a measurable unit to compute mouse movement. The IMU consists of multiple different sensors, such as accelerometers, gyroscopes, and magnetometers, which measure acceleration, angular velocity, and strength and direction of magnetic fields. This information is fundamental to retrieving data and processing precise mouse movements. The steps below summarize how the IMU data will be processed into data to be used for mouse movements:

1. Sensor Data Collection: Retrieve linear acceleration, gravity, and angular velocity data from the accelerometer, gyroscope, and magnetometer in the IMU.

2. Orientation Estimation: Apply Mahoney or Madgewick's sensor fusion algorithm to estimate the orientation of the IMU in 3D space.

3. Gravity Compensation: Extract the gravitational acceleration vector from the accelerometer data so that linear acceleration due to hand motion is separated from gravitational acceleration.

4. Linear Acceleration Calculation: Subtract the compensated gravitational acceleration from acceleration in the x, y and z directions.

5. Position Calculation: Double integrate the calculated linear acceleration in the x and y directions to get x and y position.

6. Error Correction: Over time, integration will lead to position drift due to sensor noise. The IMU we have purchased for this product comes with calibration software to help mitigate this error.

6.2.2 Mouse Movement

To control the mouse on the laptop, we will be using the PyAutoGUI library on Python. It is an important component of our product as it provides the software framework to convert hand movements detected by the IMU into mouse movements and interactions with the computer. PyAutoGUI is cross platform so it can work on various operating systems including Linux, Windows, and macOS. Specifically, it provides functions to simulate mouse actions, such as moving the mouse to specific coordinates on the screen, clicking, dragging, and scrolling. This allows the product to mimic traditional mouse behavior. In addition, customization options for movement speeds, acceleration rates, and other parameters are also available which will allow the IMU data to be integrated more seamlessly. Overall, PyAutoGUI offers cross-platform compatibility, customization options, and accessibility, making it a versatile tool for creating an intuitive and efficient user experience.

6.2.3 Bluetooth

In our Bluetooth Low Energy (BLE) client-server model, the esp32 acts as the server, posting notifications to its subscriber clients. The Python client running on the user's laptop listens for notifications and carries out the appropriate actions. For the GATT profile, we created two different characteristics - an opcode characteristic and an IMU data characteristic. When the esp32 detects a gesture, it translates this to an opcode (ex. RING, FLEX_POINTER). These opcodes, represented by an integer, are notified to the user's laptop. When the Python client receives this opcode in a callback function l, it carries out the currently mapped gesture (ex. closing a tab, increasing brightness).

For the IMU characteristic, if the esp32 detects the failsafe being triggered, it begins sending x and z acceleration data every 50 milliseconds. The data is represented as a byte array containing the bytes of two floats. These values are recovered on the client side; they are received by the IMU callback function and processed by our IMU module.

7 TEST & VALIDATION

For testing, verification & validation, we broke up our testing structure into each of the design requirements we denoted above: latency, weight, accuracy, wireless range, & battery life.

7.1 Latency

For latency, we measured the individual sensor movement for each target gesture. This includes sensor detection, or the amount of time it takes for the sensor to detect that its state has been changed, signal processing, including ADC calculations performed by the board, and any noise cancellation algorithms deployed, Bluetooth data reception - sending and receiving data - and finally gesture identification - using the information given to correctly identify that a gesture has been made, what gesture was made, and the translation into keystrokes. Since a passing output for latency would be that our gestures be recognized and carried out in less than or equal to 300ms, we started by measuring the total amount of time it takes from doing a gesture to having the keystroke implemented on the laptop. From here, we compared the results of this test to our desired requirement and measured each step of the process to see where we can cut down on time. We assumed the largest bottleneck in our system now is the Bluetooth technology. For this, we tried to speed up individual components and process more at the board level to minimize the amount of information sent over Bluetooth, and, in turn, the total latency of the system. As a backup plan for Bluetooth, we aimed to use USB or a more general serial communication approach to send data from the board to the laptop. To carry out our test for this component, we constructed a test in which we pressed start on a stopwatch with the mouse while at the same time triggering one of the sensors. In theory, this would result in the mouse clicking stop when the gesture propagates through. From here, we were able to achieve an average of 40ms of latency for the sensors (not including the IMU), which is well under our use case requirements.

7.2 Weight

For weight, we wanted the weight of our product to be as minimal and unnoticeable as an average watch, weighing in at around 113-170g. We figured that our greatest potential risk to this requirement was be the battery. For this, we again tried to minimize the weight of other components we use, including the sensors, main board, PCB, and IMU. We had to compromise battery life for the purposes of weight, but did our best to balance the two to have sufficient battery life to meet our requirements while also not being too heavy to meet our weight requirements. In order to test our weight requirements, we had a technical weight evaluation. For the weight evaluation, we simply weighed the glove on a metric scale, and recorded the values. Additionally, to find bottlenecks and ways to cut down the weight should we need to, we weighed each individual component and try to find ways to limit weight while also maintaining functionality. We recognized that although we may meet our weight requirements, user comfort is also an important aspect of our design, and we aim to satisfy this factor as well. Our final weight was 191.78g, which is sadly above our use case requirements. The battery we decided to use was a 10,000mAh 5V @2.1A portable charger that constituted 57% of the total weight of the glove. Although we had to go over our budgeted weight, a tradeoff we discovered is that our glove has a very large battery life, which, for 20g extra (which isn't much recognizable) seemed worth the tradeoff.

7.3 Accuracy

In terms of accuracy, we planned on testing our product extensively to make sure we meet our requirement of at least 90% of gestures correctly identified. Some risks we planned on seeing here are thresholding and human variance. In order to mitigate as much risk as possible, we tested our product and GUI with a user study of around 10 participants, and make a point to make wider thresholds to accommodate more variance in between users. Our failure plan for this requirement was to reduce the number of gestures as a whole, to simplify the process itself, or make the gestures more distinct, in that there would be less factors to keep track of in determining what gesture is recognized. When we first designed the placement of each sensor on the glove, we positioned each touch sensor at the tip of each finger and each flex sensor relatively high, right below the touch sensors. After a small user study of about 5 people, our accuracy was very low, with about 40% of the sensors working as promised. We saw that as people were reaching to touch the touch sensors, their fingers were bending just enough to trigger the flex sensors as well. We then decided to move the touch sensors down and to the side of their original positions, so the users won't have to bend their fingers as much to reach the touch sensors. As a quick demonstration, when you touch your fingertips with your thumb, you can see that your finger bends quite significantly. Moving the sensors to the side and down allow you to touch each sensor without too much of a bend. With proper thresholding, these slight bends will not produce a trigger. With this new positioning, we were able to achieve our use case requirements and satisfied our users. Finally, we achieved a 100% accuracy for our flex and touch sensors on the glove. As for the IMU accuracy, we measured this accuracy in terms of how many seconds it takes for a user to reach a desired position. For this test, we held a small user study of around 10 participants. We tasked each participant to use the glove to reach a button on the screen to be clicked. We repeated this process for three trials per participant For all the participants, it took an average of 10.2 seconds to reach the desired button. After the third round of trials, we discovered that the time in which participants were able to reach the button decreased slightly to an average 7.1 seconds. Although this accuracy is quite low, we recognized that there is a learning curve to the device, as well as certain movements that can be done to increase accuracy. This includes holding the glove steady when more error compounding is detected, as to zero out the velocity and error curves. With more filtering as well as user training, we can lower this accuracy even further.

7.4 Wireless Range

For our wireless range, we wanted our minimum distance to be around 2.28 meters, or the average distance between the couch and a TV. In order to test our device for this range, we conducted a test where we signaled a gesture less than 1 foot in front of the laptop, and recorded if it carries out the gesture or not. From here, we repeated the same process of signaling a gesture, while backing up 1 foot in between each gesture. One of the major obstacles to achieving this range is the actual Bluetooth networking. We believe there will be lots of noise and interference for our Bluetooth device, so we will try to mitigate this as much as possible through noise dampener algorithms and reducing the number of packets sent in total. If we are unable to meet this requirement we will resort to using a serial interface such as USB to communicate between the board and the laptop. After testing, we were able to achieve a Bluetooth range of 3.05 meters in all directions, which surpasses our use case requirement of 2.28 meters.

7.5 Battery

For the battery life of our device, we aimed to have a battery life of greater than or equal to 2-3 hours, which is the average length of a movie. We planned to measure the time it takes for our device to go from fully charged to completely dead/off. For the test itself, we mimicked scenarios in which users will most likely use our glove. Specifically, we tested out gestures on the glove every 5-7 minutes until the battery ran out. Additionally, we looked through data sheets to find out which devices have the highest battery consumption and try to mitigate power consumed by other devices based on this information. The board and IMU ended up taking up the most battery life, so we used our board's low power mode during idle states, and the enable pin for the IMU also during idle states. We believed this would provide the most useful information, as we expect our users to use the glove for only a couple minutes at a time, sporadically throughout the course of a movie. Some risks to this requirement include power requirements per sensor/device, and sensor integrity. From learning how lithium batteries work, we know that the voltage output of the battery will reduce over time, regardless of battery life remaining. Since our battery voltage will be relatively close to the typical voltage inputted to the device (3.3 typical inputted vs. 3.7-5V VIN), we mitigated this risk by assuming 85% of battery life for our baseline, since voltages tend to dip more significantly during the last 10% of battery life. If we fail to meet the battery life requirements, we will need to use a larger battery, with a higher voltage rating. However, this failure plan must still be on track with our weight requirement. Since the two are closely linked (larger battery = more weight), we will try to balance these two requirements so both are satisfied at any given time. As mentioned previously, the results of this tests produced a battery life of around 12 hours. This is greatly due to the fact that we are using a larger battery pack of 10,000 mAh with 5V @2.1A, as well as the utilization of Bluetooth Low Energy and our IMU device's low power mode. Although the PCB has the capability of receiving power through a lithium polymer battery, we decided that a portable charger was the easiest way for users to understand and charge their batteries, therefore satisfying our users as well as our design requirements.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in Fig. 7 is our Gantt chart for the specific task breakdown of our project. Yellow is Sarah, Red is Rosina, Blue is Saumya, Purple is Rosina & Saumya, Orange is Sarah & Rosina, and Pink is all. The project was mostly on schedule, except for a few components. The parts that took longer than we anticipated were the IMU data processing and fabricating the PCB board. We ran into unexpected problems that made it difficult to adhere to the schedule. However, the PCB board was still done in a reasonable time and the IMU processing was done by the final presentation, although more time would have allowed us to make the mouse movements more precise.

8.2 Team Member Responsibilities

In terms of the overall project breakdown, Sarah is responsible for implementing and testing our hardware, interfacing between hardware and software, calibrating sensors, and helping with the physical components of our design. Saumya is responsible for the low-level code running on the microcontroller to send data to the Python receiver, Bluetooth networking, and sending pruned sensor data over to the pyautogui-based mouse and keyboard module. Rosina will be responsible for sensor value processing for keystrokes and the IMU (with help from Saumya), the mouse and keyboard module, and the GUI of the system.

8.3 Risk Management

Throughout the project lifecycle, the main risks we encountered included the PCB and the IMU. As for the PCB, as mentioned before, we explored two options: a milled PCB as well as ordering a custom PCB online. Since ordering a PCB would produce much smaller traces and in return a much smaller board in area, we decided to go with ordering our PCB. After reviewing our budget, we had more than enough left to spend on boards, and decided to use the milled PCB as our fallback design. Additionally, as for the IMU, we had lots of trouble regarding signal processing and converting the acceleration data to mouse movements. Because our IMU unfortunately arrived quite late in the semester, we mitigated this shipping risk as much as we could by figuring out integration procedures prior to arrival and generating pseudo code for how the code would analyze data from the IMU. This helped greatly as once the board arrived we were able to start right at the mouse movements stage. Again, because of time constraints, we tried to mitigate the compounding error as much as possible by using a Kalman filter. This helped a lot with the compounding error as well as any noise generated from the IMU itself.

8.4 Bill of Materials and Budget

The bill of materials and budget needed for the project are listed in Table 1.

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Main development board	ESP-WROOM-32	Espressif Systems	1	$9.59	$38.36
Intertial Measurement Unit	FSM300	CEVA Technologies, Inc.	1	$67.36	$67.36
Touch Sensors	N/A	Ezweiji	6	$8.19	$49.14
Flex Sensors	SEN-10264	Sparkfun	5	$17.09	$85.45
Fabric Glove	N/A	HandLandy	1	$13.80	$13.80
Printed Circuit Board	N/A	JLCPCB	10	$1.21	$12.10
10K Ohm 0805 Resistors	N/A	Chanzon	100	$4.99	$4.99
2.2K Ohm 0805 Resistors	N/A	Chanzon	100	$4.99	$4.99
Total					$276.19

8.5 Risk Mitigation Plans

As previously discussed, we have several risk mitigation strategies for each of our use case requirements. For latency, if we are unable to get below our threshold, we plan to switch from Bluetooth to USB as our protocol for sending data between the glove and the computer. For weight, we will opt for a smaller battery in the case that we cannot reduce our weight in any other component to get below our threshold. For accuracy, if it becomes impossible to achieve our goal, we plan to reduce the complexity of our gesture language and/or make our gestures more distinct. For wireless range, if we are unable to reduce interference, we may switch to either WiFi or USB, since the ESP32 also supports WiFi. Finally, for battery life, in the case that we are not able to power our device with a 5V battery, we will increase the size of our battery.

9 ETHICAL ISSUES

There are a few ethical issues that arise in our product. First, our glove is a right handed glove. This negatively affects those that are left-handed, as they will not be able to use the glove as efficiently as a right-handed person would. In theory, this could be fixed by making a duplicate product that functions the same way, but uses a left-handed glove. Another issue is the exposed circuitry on the glove. There is no encasing on the microcontroller, sensors, or wires, which could potentially lead to easier damage and physical harm to the user. To fix this issue, we could put the exposed components in a protective casing. Finally, a large ethical issue that relates to our product is privacy. The user should be fully aware of the extent to which the glove can be used. This includes possibly deleting data, or downloading software that may be harmful. Additionally, if this glove were ever customized using keystrokes the user is not aware of, it might cause adverse effects to the user's data and privacy. To solve this issue, we can introduce password protection in order to change the keystrokes on the glove through the GUI. This might allow for a safer product for the user as well as make sure their information and privacy is secured.

10 RELATED WORK

Some other related works that are similar to our product is gest.co, the Nintendo Power Glove, and various ASL interpreter gloves. The Gest glove leverages Bluetooth Low Energy technology and a similar glove sensor system that can track gestures from afar. While Gest is still in its prototype stages, it provided us a good baseline for existing projects in the space. The Nintendo Power Glove is a controller accessory for the NES (Nintendo Entertainment System). It provided users a way to control video games on their console straight from their hands, rather than a traditional controller. There are various buttons and controls located on the glove, where users can control players and use their hand motions to control characters. Additionally, similar to our design, the Power Glove detects yaw, pitch and roll to detect hand placement and orientation relative to the origin. As for the ASL glove, we found a previous Capstone project team that created an ASL interpreter glove that could detect hand movement for the entire ASL alphabet. Their system used a similar style of wearable technology fit with sensors, which we used as inspiration for our sensor setup.

11 SUMMARY

In summary, we've successfully developed a functioning glove capable of translating gestures into keystrokes and facilitating mouse movements. However, we encountered unexpected challenges, notably concerning undesired mouse movements and the integration of a fail-safe mechanism. While the system doesn't operate as seamlessly as initially intended, these challenges have provided valuable insights for potential improvements in future iterations. Moving forward, we plan to explore alternative positioning systems to replace the IMU to minimize compounding errors. These solutions could potentially entail computer vision.

11.1 Lessons Learned

To future students, we would suggest heavily researching parts to use that provide more functionality. Something that was particularly helpful to us was purchasing an IMU that had preinstalled calibration software. This simplified the processing a great amount and saved us a lot of time. Externally sourcing a PCB instead of making our own also saved us time and allowed us to avoid any possible errors during fabrication. Finally, our last suggestion would be to look into other sensors that can track position more accurately than an IMU. Position could not be accurately derived from acceleration, so if the goal is to find accurate positions, we would suggest looking into other softwares/sensors.

12 GLOSSARY OF ACRONYMS

ADC – Analog to Digital Converter
BLE – Bluetooth Low Energy
DAC – Digital to Analog Converter
ESP – ESP-WROOM-32 Microcontroller
GUI – Graphical User Interface
IMU – Inertial Measurement Unit
PCB – Printed Circuit Board

13 REFERENCES

[1] Espressif Systems. "ESP32-WROOM-32 Datasheet." 13 Feb. 2023, https://www.espressif.com/sites/ default/files/documentation/ esp32wroom32_datasheet_en.pdf

[2] CEVA, Inc. "FSM30x Datasheet." 2 Jan. 2023, https://www.ceva-dsp.com/wp-content/uploads/2019/10/FSM30x-Datasheet.pdf.

[3] "Ezweiji Precision Electrical Resistance Touch Sensor (Resistive)." Amazon, https://www.amazon.com/Ezweiji-Precision-Electrical- Resistance-Resistive/dp/B0C1SF7JT5 /ref=sr_1_1?crid=34RFCPF23WL3U& keywords=ezweiji+touch+ sensor&qid=1697250980& sprefix=ezwei ji+touch+sensor%2Caps%2C101&sr=8-1.

[4] Sparkfun. "FLEX SENSOR SPECIAL EDITION DATA SHEET."n.d, https://cdn. sparkfun.com/assets /9/5/b/f/7/FLEX_SENSOR _-_SPECIAL_ EDITION_DATA_SHEET _v2019__Rev_A_.pdf.

[5] Alex. (n.d.). Using PyKalman on Raw Acceleration Data to Calculate Position. Stack Overflow. Retrieved December 15, 2023, from https://stackoverflow.com/questions/47210512/using-pykalman-on-raw-acceleration-data-to-calculate-position

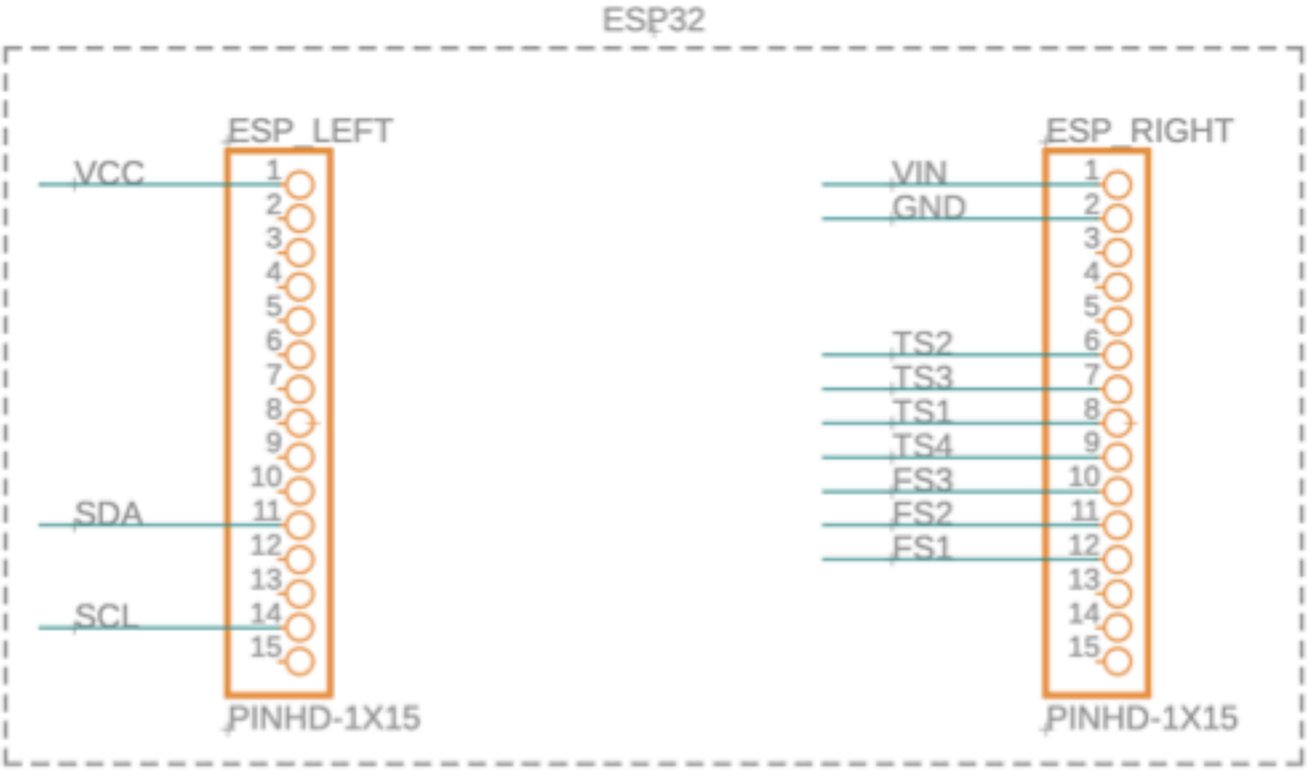


Figure 1: A schematic picturing the pins and connections for the ESP32 development board.

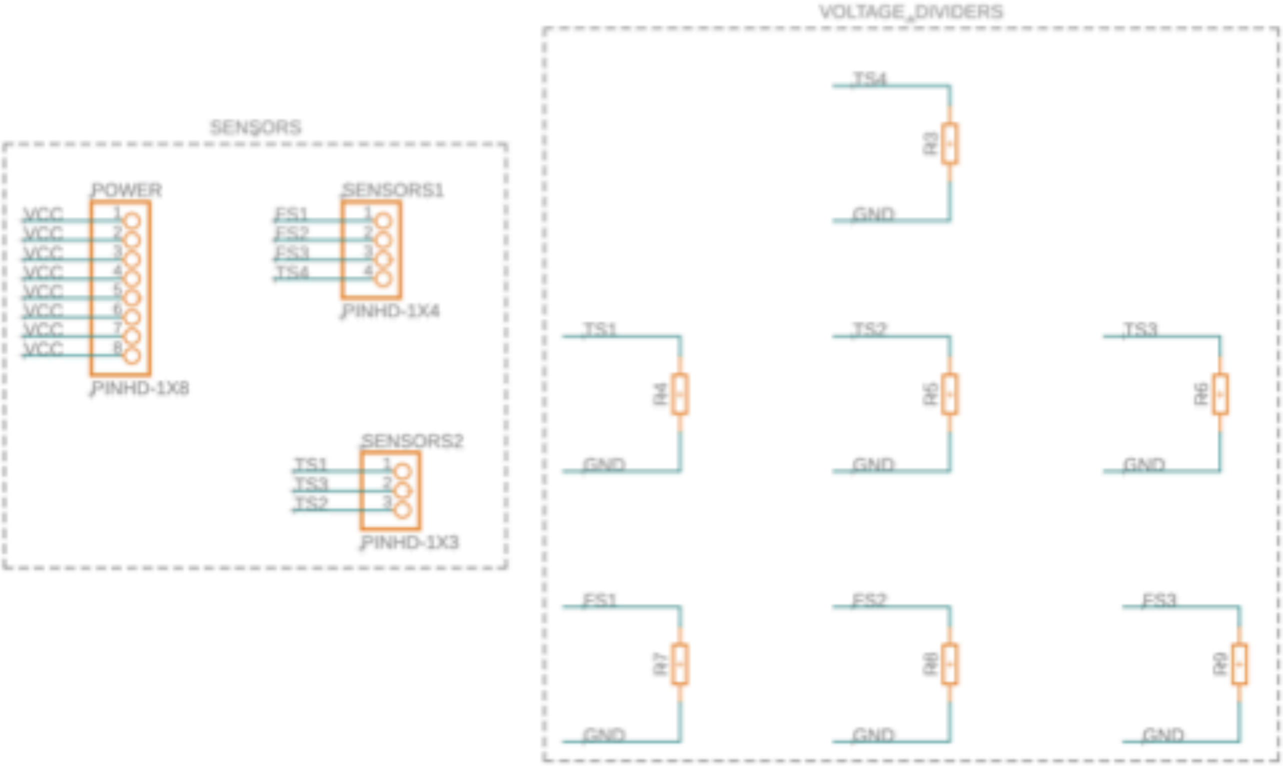


Figure 2: A schematic picturing the pins and connections for each sensor in the project.

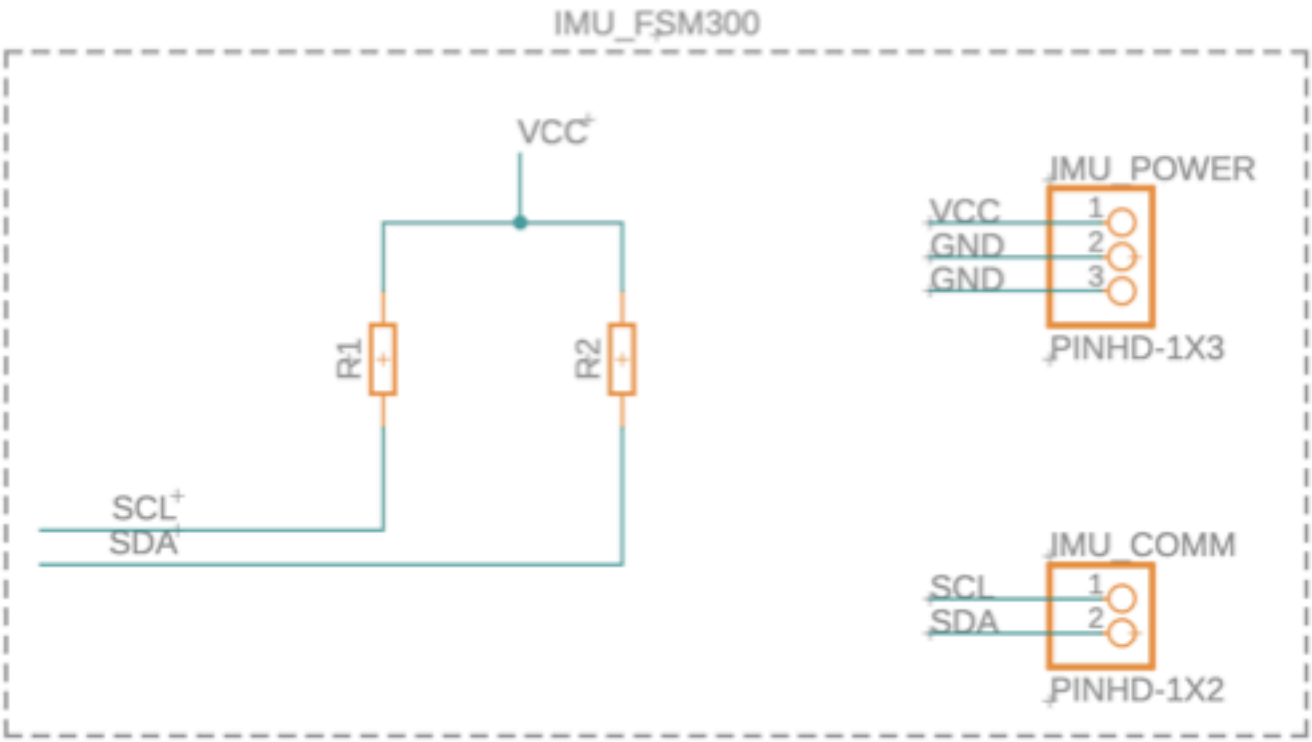


Figure 3: A schematic picturing the pins and connections for the IMU.

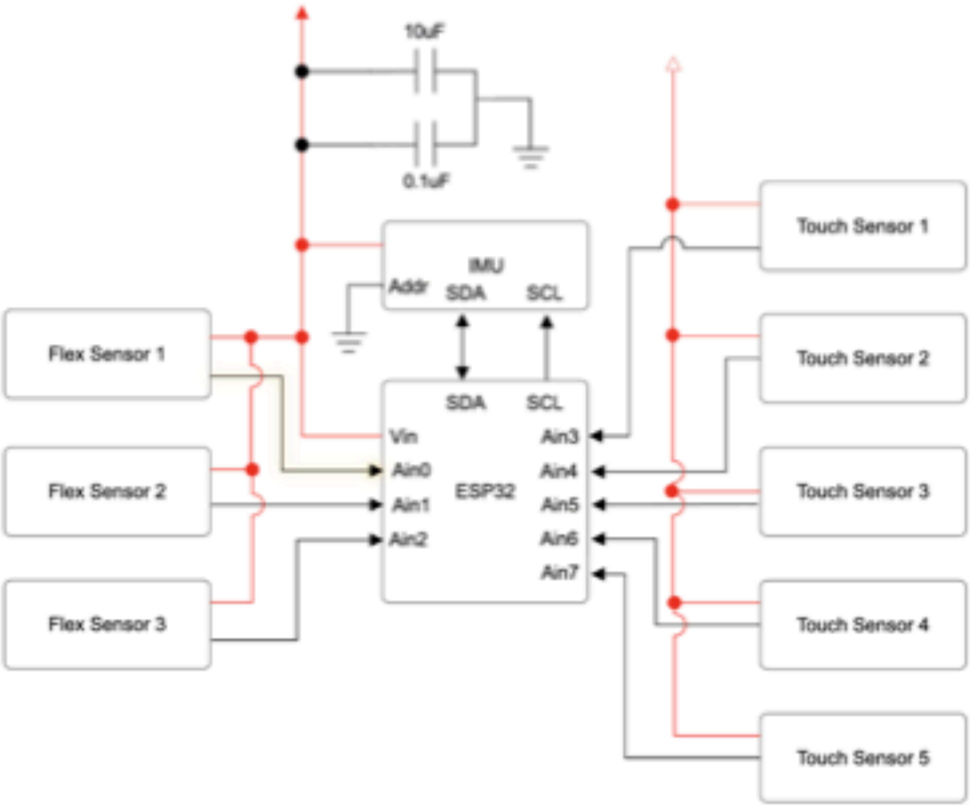


Figure 4: A block diagram depicting the hardware approach to the project.

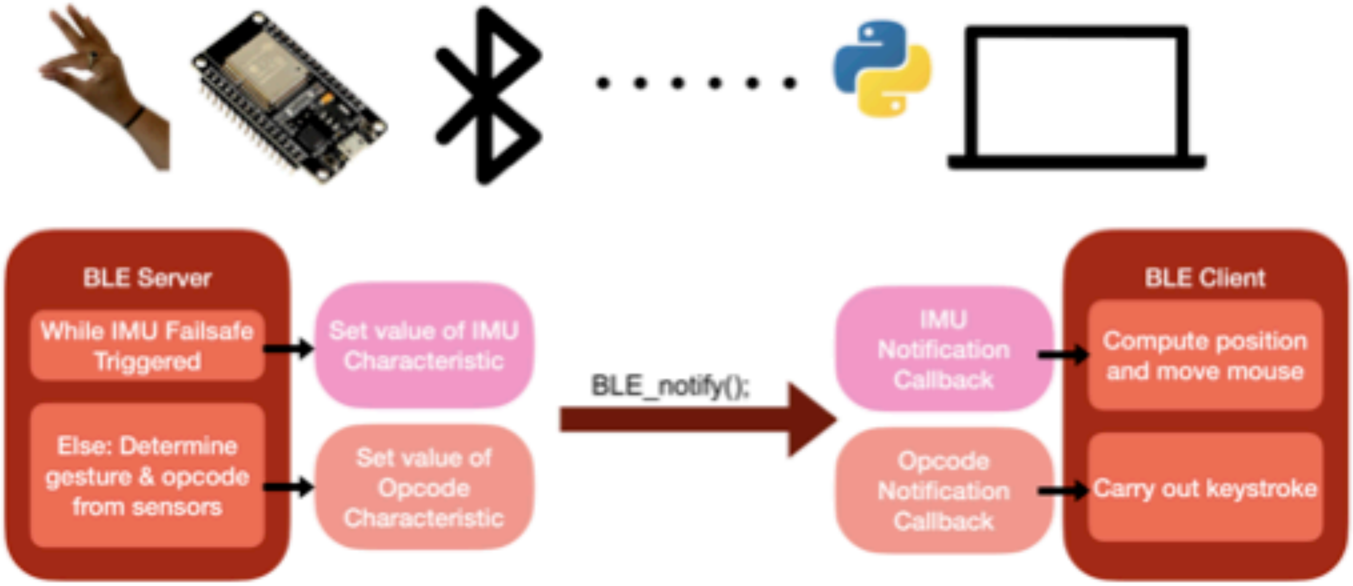


Figure 5: A block diagram depicting the Bluetooth approach to the project.

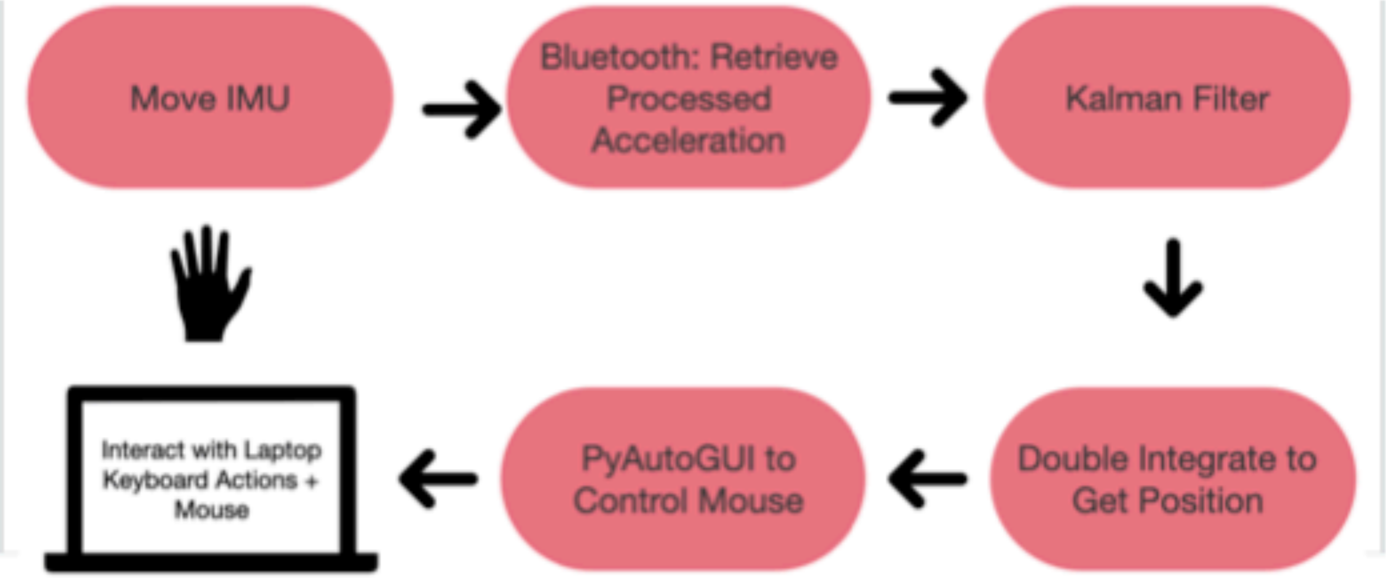


Figure 6: A block diagram depicting the IMU design flow for the project.

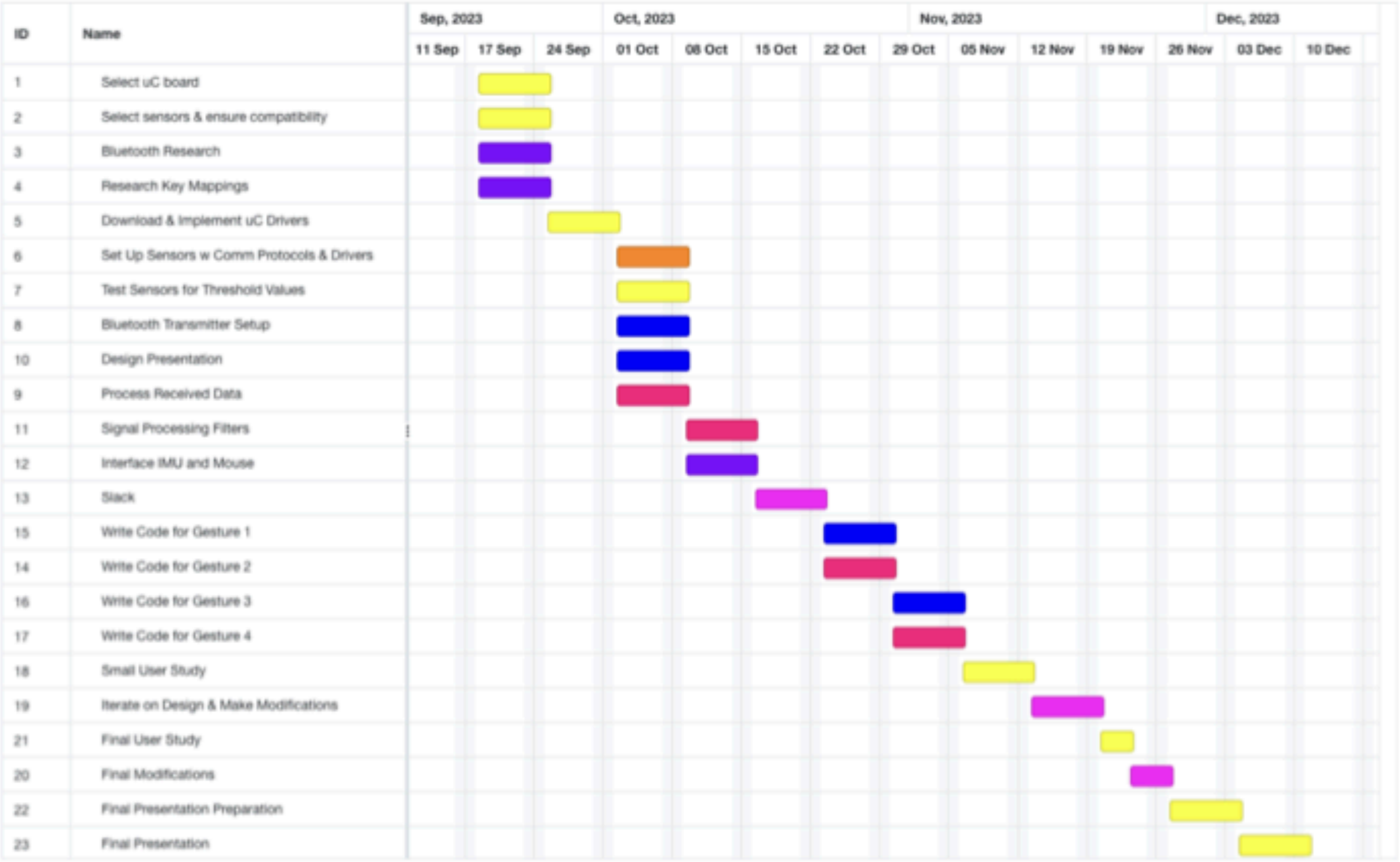


Figure 7: Gantt Chart